ZG: An Alternative Way to Build Simple GUI Applications

Billy Chang
ThinkingCactus Ltd.
billy.chang@thinkingcactus.com

Abstract

The paper presents a novel way of building GUI applications, in a more designer-oriented style GUI programming, as oppose to the common OMG-I-Don't-Want-To-Be-The-Person-That-Maintains-This-Code sort of GUI programming. The paper primarily deals with programming for desktop software.

1. Introduction

Python is a great language. It let you do things you want to do without getting in the way of you doing it too much. Python code are very straight forward to write and maintain. Like cute furry puppies, coding in Python makes you feel the world is a better place.

Except, when you're writing GUI layout code, that's when python code starts to look like evil C++ code. Ewww.

1.1. How It Used To Be

Typically in Python there are two ways of building GUI: you either go with the straight coding approach, making individual function calls to place individual UI elements; or you go with the (X)HTML + CSS + bitmaps approach, much like how most web pages are built. We discuss them a little bit here.

1.1.1. Straight Coding

This is the mainstream way to code for desktop software, a practise inherited from the procedural programming days, instruction-by-instruction the GUI is built using tables or sizers (wx-terminology), fitting individual buttons and textfields into them, setting properties for each of these along the way. Even as python programs, it takes an uncharacteristically large amount of effort to get working, hence developers are generally advised to use GUI designers, which are just barely adequate for GUI creation but are not good for maintenance. Maintenance process requires executing code often after every change to check if GUI looks ok, since it's not WYSIWYG. Even a relatively simple application will haunt the programmer with large amount of unmaintainable cruft.

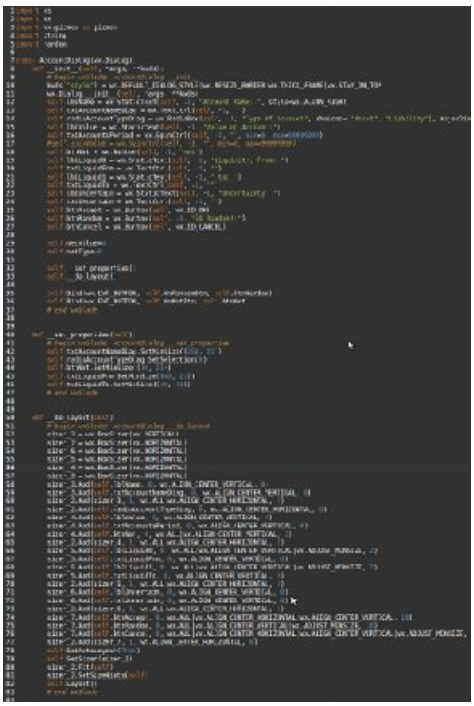

Figure 1.1.1.1 Example of GUI layout code, 83 lines of code.

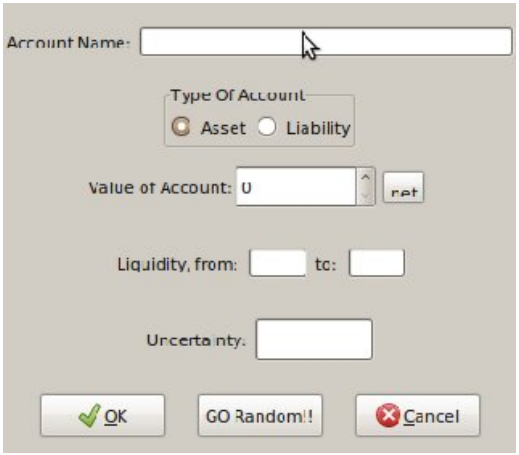

Figure 1.1.1.2 What the above code created.

The code show in Figure 1.1.1.1 shows the layout code required for a relatively simple GUI, as seen in Figure 1.1.1.2. Note, the code shown are only the code used for laying out the GUI elements for the GUI shown, the business logic and initialisation boiler plates were omitted.
Characteristics for this type of code is that, it can be relatively easy build tests for, since everything is code, and not too bad for code coverage, but very hard to debug and thus maintain, not to mention the tendency to puts the developer in a foul mood when he/she realises 70% of the code just written were unreadable and for purpose of layout only.

GUI designers may help a little during the creation process, but being automatically generated code they can be extremely difficult to change and even more difficult to debug.

1.1.3. (X)HTML + CSS + bitmaps

Being able to layout web page elements nicely using HTML developed some people thought to do the same for desktop applications, but other than software that were designed to be 'theme-able', or as plug-ins to other systems, it was still generally quite a rare approach. And because it is rare, most engines out there of this type were custom-built for their specific applications only. The most well-known example of this might actually be Firefox addons.

From a maintainability point of view, this is generally better than the straight coding approach, and easily theme-able just by changing to a different HTML or CSS is a plus. But the download side however, is that developers will end up with a large number of files to manage, mostly bitmaps.

1.2. ZG: An Alternative

This paper presents a user interface library that was developed to address these problems, where the separation of the design (view) and business logic (controller) means that it is relatively simple to maintain, easily theme-able, but also by design easy to manage. This is achieved using a vector graphics format, namely .svg. A single window application, is one .svg file for GUI design, and one .py file for business logic.

1.3. This is NOT New

No, this idea isn't new, Adobe (Macromedia) Flash had been doing this for a while, so it is agreed that this isn't a brand new idea. If anything, SVG specification have user interfaces specifications inside, so it even seemed like it was fully the intention for someone to do this.
So, this is not new, just different. Not designed by committee, but by me, in a pragmatic way, with a lack of documentation and obviously far from complete.

2. ZG (Zed-Gee)

Originally named 'ZenGarden', the GUI library aspires to be 'A Place of Peace', it had since shortened its name to ZG.

2.1. Design Philosophy

The way I see it, 'Designer != Programmer', with designer being the one in control of the experience of what the user go thought, while not making the work too difficult for the programmer.

Beside this, I also want this to be:

- As open source as possible
- Flexible enough to be able to add different attributes to the GUI elements easily

The .svg, an open format, satisfies both the above points, and more. In an .svg, everything can have its own name, its own arbitrary set of attributes and values, while stay a single text file that's good for version control and compression, makes this an ideal format to use. Being an XML format doesn't hurt either since good parsers are easy to find.

2.2. Tools and Dependencies

2.2.1. Inkscape

Inkscape is currently the primary tool for creating .svg files for ZG, itself being a very capable open source vector graphics drawing system, plus it has a XML editor which can be used to directly change the .svg, makes it ideal.

2.2.2. Enthought Enable

Built on top of wxpython, Enable utilises agg (Anti-Grain Geometry) as the rasterisation engine on both Windows and Linax, and uses Quartz on Mac, thus in theory zg can be ported to these platforms with relative ease. Alternatives to agg, cairo is a possible candidate since community is a lot more active, although cairo is a lot slower than agg, and the anti-aliasing not as pretty.

2.3. Mechanism

SVG files created for ZG have elements (called zgelements) with names in a specific format, which specify its zg-name, its type, and its behaviour regarding window resizing. Depending on the type, there can be further type-specific attributes/values inserted onto the element in the .svg file. Majority of the behaviour of these zgelements are specified in the .svg files too, such as the change in visual elements on mouse overs / toggle buttons, the use of different cursors and fonts for a textfield, are all part of the .svg file and never touches the python code. In other words, the general visual experiences, i.e. the designer's responsibilities, all resides in the .svg file.

The python code that resides in the corresponding .py file, besides specifying which .svg to load, only need to connect zgelements by name in the .svg.

On occasions when a unconventional custom GUI element needs to be built, class inheritance by code can also be done.

Only on occasions that the developer WANTS to change the layout themselves, do layout code ever appear. There are no pictures to show an equivalent code in ZG as Figure 1.1.1.1, since there are NO layout code in python, as all positioning and resizing information are in the .svg file.

2.4. Real-World Use

ZG was built as part as the user interface for ThinkingCactus' flagship production, ZenCub3d, an easy to use 3D animation software system. And because we're trying to push the boundary of what's considered to be easy to use, a good mix of conventional and non-conventional user interface elements needed to be developed, and ZG has proved to be able to handle these successfully.

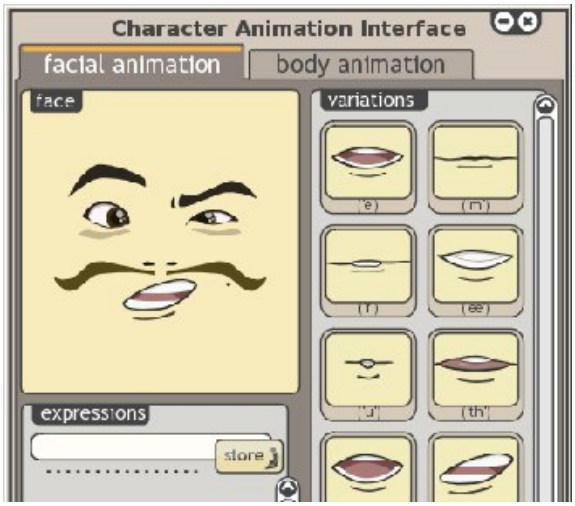

Figure 2.4.1 Complex User Interface Can Be Achieved with Style.

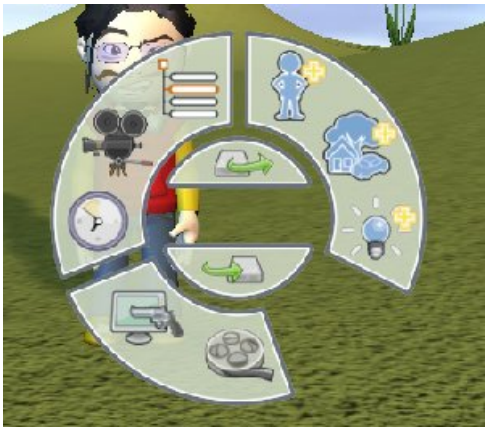

Figure 2.4.2 Integration of ZG GUI into the ZenCub3d.

It can be seem in Figure 2.4.2, ZG generated the semi-transparent radial mouse menu that's integrated into the 3D system, as well as generating the vector graphics 2D face texture on the character's face, from the user interface as seen in Figure 2.4.1.

2.6. Current Limitations

ZG, being built so far in a purely pragmatic approach, within a small team that have way too many tight deadlines, feature requests are rejected unless absolutely crucial, therefore it still currently have many limitations. For example, ZG currently can not do more advanced matrix-based geometry operations, such as vector rotation and scaling. Gradient colouring and pattern fill is also currently missing.

2.7. Future Directions

ZG is currently closed sourced, but it is the author's intention to have it be open source someday, just not having the resources to fully support it being open at the mean time. Licensing being an issue, if ZG becomes open source, the intention is to use the BSD license, as used by the Enthought family of libraries.

As further development goes on, it will likely be ported to use different GUI system engines, possibilities being clutter, Qt (via PySide), if cairo's glitz optimization through OpenGL gets fast, it's a possibility too.

3. References

SVG Specification: http://www.w3.org/TR/SVG/

Inkscape: http://inkscape.org

Enthought Enable: http://code.enthought.com/projects/enable/

Clutter: http://www.clutter-project.org/

Qt: http://qt.nokia.com/

PySide: http://www.pyside.org/

Cairo: http://cairographics.org/